On the Quality of Relational Database Schemas in Open-source Software

Fabien Coelho, Alexandre Aillos, Samuel Pilot, and Shamil Valeev
CRI, Mathématiques et Systèmes, MINES ParisTech,
35, rue Saint Honoré, 77305 Fontainebleau cedex, France.
fabien.coelho@mines-paristech.fr, *firstname*.*lastname*@mines-paris.org

***Abstract*—The relational schemas of 512 open-source projects storing their data in MySQL or PostgreSQL databases are investigated by querying the standard *information schema*, looking for overall design issues. The set of SQL queries used in our research is released as the *Salix* free software. As it is fully relational and relies on standards, it may be installed in any compliant database to help improve schemas. Our research shows that the overall quality of the surveyed schemas is poor: a majority of projects have at least one table without any primary key or unique constraint to identify a tuple; data security features such as referential integrity or transactional back-ends are hardly used; projects that advertise supporting both databases often have missing tables or attributes. PostgreSQL projects appear to be of higher quality than MySQL projects, and have been updated more recently, suggesting a more active maintenance. This is even better for projects with PostgreSQL-only support. However, the quality difference between both databases management systems is mostly due to MySQL-specific issues. An overall predictor of bad database quality is that a project chooses MySQL or PHP, while good design is found with PostgreSQL and Java. The few declared constraints allow to detect latent bugs, that are worth fixing: more declarations would certainly help unveil more bugs. Our survey also suggests that some features of MySQL and PostgreSQL are particularly error-prone. This first survey on the quality of relational schemas in open-source software provides a unique insight in the data engineering practice of these projects.**

***Keywords*-open-source software; database quality survey; automatic schema analysis; relational model; SQL.**

I. Introduction

This paper is an extended version of *A Field Analysis of Relational Database Schemas in Open-source Software* [1] presented at DBKDA 2011. Compared to this initial version, 512 schemas are surveyed instead of 407, which enhances the accuracy of the statistical validation of our analyses; the maintenance status of the surveyed projects was collected again as of January 2012; comments have been updated and added to reflect the new data; more detailed tables are provided about the results; the bibliography is much more thorough, with over 50 new references; an appendix describes the advices available with our schema analyzer; the paper page count, excluding the appendix, is increased from 7 to 10 pages.

In the beginning of the computer age, software was freely available, and money was derived from hardware only [2]. Then in the 70s it was *unbundled* and sold separately in closed proprietary form. Stallman initiated the free software movement, in 1983 with the *GNU Project* [3], and later the *Free Software Foundation* [4], which is now quite large [5][6] and expanding [7] (Predicts 2010) to implement his principle of sharing software. Such free software is distributed under a variety of licenses [8], which discuss copyright and liability. The common ground is that it must be available as source code to allow its study, change and improvement as opposed to compiled or obfuscated, hence the expression *open source* [9][10][11], This induces many technical, economical, legal, and philosophical issues. Open-source software (OSS) is a subject of academic studies [12] in psychology, sociology, economics, or software engineering, including quantitative surveys. Developers' motivation [13][14][15][16][17], but also organization [18][19][20][21][22][23][24][25][26] and profiles [27][28][29] are investigated, as well as user communities [30]; Existing economic frameworks [31] are used to analyze the phenomenon, as well as the influence of public policies [32]. Research focusing on software engineering issues can also be found. The development of the Apache web server popular [33] is compared to non-OSS projects [34] and its user assistance is analyzed [35]. Quantitative studies exist about code quality in OSS [36][37][38][39][40] and its dual, static analysis to uncover bugs [41][42]. Database surveys are available about market shares [43], or server exposure security issues [44]. This study is the first survey on the quality of relational database schemas in OSS. It provides a unique insight in the data engineering practice of these projects.

Codd's relational model [45] is an extension of the set theory to relations (tables) with attributes (columns) in which tuple elements are stored (rows). Elements are identified by keys, which can be used by tuples to reference one another between relations. The relational model is sound, as all questions (in the model) have corresponding practical answers and vice versa: the tuple relational calculus describes questions, and the mathematically equivalent relational algebra provides their answers. It is efficiently implemented by many commercial and open-source software such as Oracle, DB2 or SQLite. The *Structured Query Language* (SQL [46]) is available with most relational database systems, although the detailed syntax often differs in subtle and incompatible ways. The standardization effort also includes the *information schema* [47], which provides metadata about the schemas of databases through relations.

The underlying assumption of our study is that applications store precious transactional user data, thus should be kept consistent, non redundant, and easy to understand. We think that

database features such as key declarations, referential integrity and transaction support help achieve these goals. In order to evaluate the use of database features in open-source software, and to detect possible design or implementation errors, we have developed a tool to analyze automatically the database structure of an application by querying its *information schema* and generating a report, and we have applied it to 512 open-source projects. The notion of the quality of a database schema design is quite elusive, as shown in Burkett's overview [48], with a lot of focus on qualitative assessments. Key criteria such as understandability, simplicity, expressiveness, maintainability or evolvability are hard to transform into basic objective metrics. A review process has been proposed to evaluate the quality of relational schemas [49], at the price of mostly manual investigations by field experts. Some quality focus on the conceptual schema and compare alternative models [50][51] by recognizing patterns. Following MacCabe's metric to measure automatically program complexities [52][53][54], several metrics address data models [55][56] or database schemata either in the relational [57][58] or object relational [59] models, including experimental validations [60]. These metrics rely on information not necessarily available from the database concrete schemas. Moreover, such approach help compare two schemas that model the same application domain, but are less useful when used about unrelated schemas. We have rather followed the dual and pragmatic approach [61], which is not to try to do an absolute and definite measure of the schema, but rather to uncover issues based on static analyses. Thus, the measure is relative to the analyses performed and results change when more are added. Static analysis on user application codes (not simply the schema) could also be used to help uncover hidden constraints in a schema (for instance, a join between two tables suggests a possible foreign key) and to use them to improve data quality [62], but this is beyond our simple approach.

The remainder of this paper is organized as follows: Section II presents the methodology used in this study. We describe our tool, our rating strategy and the statistical validation used on the assertions derived from our analyses; Section III lists the projects by category and technology, and discusses similarities and differences depending on whether they run on MySQL or PostgreSQL; Section IV describes the results of our survey, with quite a poor overall quality of projects, as very few database schemas do not raise error-rated advices; Section V gives our conclusive thoughts.

II. Methodology

Our *Salix* automatic analyzer [63], is based on the *information schema* provided by standard databases. It is open-source, and its schema itself is included in this survey. In this Section, we discuss the queries, then describe the available advices, before presenting the statistical validation used.

A. Information schema queries

Our analyses are performed automatically by SQL queries on the databases metadata using the standard *information schema*. This relational schema stores information about the databases structure, including catalogs, schemas, tables, attributes, types, constraints, roles, permissions, etc. The set of SQL queries used for this study are released as the *Salix* free software. It is based on `pg-advisor` [64], a PostgreSQL-specific proof of concept prototype developed in 2004. Some checks are inspired by Currier [65], Baron [66] and Berkus [67] or similar to Boehm [68]. Note that the aim is quite different from tools which focus on advising database administrators, for instance about index creation [69]. *Salix* creates specific tables for each advice by querying the *information schema*, and then aggregates the results in summary tables in a dedicated schema. It is fully relational in its conception [70]; there is no programming other than SQL queries, but a small shell driver which *create*s the advices, *show*s or *report*s them in some detail to the interested user, and finally *drop*s them out of the database. Because of performance issues when querying heavily metadata relations, the tool relies on tables which are materialized views, although using views directly would have been a preferred option if possible. The development of *Salix* uncovered multiple issues with both implementations of the *information schema*.

B. Advice classification and project grading

The 47 issues reported by our SQL queries from the standard *information schema* are named ***advices***, as the user is free to ignore them. Although the performed checks are basic and syntactic, we think that they reflect the quality of the schemas. For instance, style advices help with understandability, and consistency advices help with maintainability. A detailed list of advices currently implemented in our tool is available [71]. Each advice has a category (19 design, 13 style, 6 consistency, 4 version, 5 system), a severity (7 errors, 21 warnings, 14 notices, 5 informations), and a level (1 raised per database, 10 per schema, 27 per relation, 7 per attribute, 2 per role). The severity classification is arbitrary and must be evaluated critically by the recipient: most of them should be dealt with, but in some cases they may be justifiable. For instance, having a mix of MySQL back-end engines is considered inconsistent and tagged as an error, although it may be necessary to do so because some features (*e.g.* full-text indexes) are only available with some back-ends. Moreover, detected errors do not imply that the application is not fully functional from a user perspective.

The 19 **design** advices focus on detecting design errors from the information available in the metadata. Obviously, semantic error, say an attribute is in the wrong relation, cannot be guessed without understanding the application and thus are out of reach of our automatic analysis. We rather focus on primary and foreign key declarations, or warn if they are missing. The rate of non-null attributes is also checked, with the underlying assumption from our experience that most data are mandatory in a relation. We also check the number of attributes so as to detect a possible insufficient conception effort.

The 13 **style** advices focus on relation and attribute names. Whether a name is significant in the context cannot be checked, so we simply look at their length. Short names are discouraged as they would rather be used as aliases in

queries, with the exception of `id` and `pk` which are accepted as attributes. We also check that the same name does not represent differently typed data, to avoid confusing the user.

The 6 **consistency** advices checks for type and schema consistency in a project, such as type mismatches between a foreign key and the referenced key. As databases may also implements some of these checks, it is possible that some cases cannot be triggered.

The 4 **version** advices focus on database-specific checks, such as capabilities and transaction support, as well as homogeneous choices of back-end engines in a project. This category could also check the actual version of a database used looking for known bugs or obsolescence. Only MySQL-specific version advices are currently implemented.

Finally, the 5 **system** advices, some of which PostgreSQL-specific, check for weak passwords, and key and index issues.

These advices aim at helping the schema developer to improve its relational design. We also use them in our survey to grade projects with a mark from 0 to 10, computed by removing points each time an advice is raised, taking more points if the severity is high, and flooring the result to avoid negative grades. The grading process is normalized using the number of possible occurrences, so that larger projects do not receive lower marks just because of the likelihood of having more issues for their size. Also, points are not removed twice for the same issue: for instance, if a project does not have a single foreign key, the same issue will not be raised again on every tables. Advices not relevant to our open-source database schema survey, *e.g.*, weak password checks, were deactivated.

C. Survey statistical validation

The data collected suggest the influence of some parameters on others. These results deal with general facts about the projects (say foreign keys are more often used with PostgreSQL) or about their grading (say MySQL projects get lower marks). In order to determine significant influences, we applied Pearson's chi-square tests [72] to compute probabilistic degrees of certainty. Beware that these statistical validations hold for our data set only. It is possible that some unwanted bias in the project selection process makes statements that are in reality false appear true, and vice versa. We followed a one project one vote principle in our analyses, so that these validations do not take into account the projects sizes or popularity. Also, our software, as all software, may include bugs with unexpected consequences. Each checked assertion is labeled with an expression indicating the degree of certainty of the influence of one parameter on an other:

very sure The probability is 1% or less to get a result as or more remote from the average. Thus we conclude that there is an influence, with a very high degree of certainty.

rather sure The probability of getting such a result is between 1% and 5% (the usual statistical threshold). Thus there is an influence, with a high degree of certainty.

marginally sure The probability is between 5% and 25%: such a result may have been obtained even if there is no influence. The statement must be taken with a pinch of salt.

not sure The probability is over 25%, or there is not enough available data to compute it. The test cannot assert that there is a significant influence. Obviously, no such assertion was included in this survey.

The rational for choosing Pearson's chi-square test is that it does not make any assumption about the distribution of values. However, it is crude, and possibly interesting and somehow true results may not be validated. Moreover, the test requires a minimal population, which is not easily reached on our small data set especially when criteria are crossed. Finally, it needs to define distinct populations: for grades or sizes, these populations are cut at the median value in order to perform the test on balanced partitions.

We also computed a correlation matrix to look for possible inter-parameter influence. The result suggested that the parameters are pretty independent beyond the obvious links (say the use of a non-transactional back-end is correlated with isolated tables), and did no help uncover significant new facts.

III. Projects

We discuss the projects considered in this study, grouped by categories, technologies, sizes and release dates. We first present how projects were selected, and then an overview.

A. Project selection

We have downloaded 512 open-source projects starting in the first semester of 2008, adding to our comparison about every project that uses either MySQL [73] or PostgreSQL [74] that we could find and install with reasonable time and effort. The database schemas included in this study are derived from a dump of the database after installation, or from the creation statements when found in the sources. These projects were discovered from various sources: lists and comparisons of software on Wikipedia (Software lists about: photo galleries, content management systems, Internet forums, reference management, issue tracking systems, wikis, social networking, church management, student information systems, accounting, weblog, Internet relay chat, health-care, genealogy, etc.) and other sites; package dependencies from Linux distributions such as Debian [75] or Ubuntu [76] requiring databases; security advisories mentioning SQL [77]; searches on SourceForge [78] which use SQL databases.

Some projects were fixed manually because of various issues, such as: the handling of double-dash comments by MySQL, attribute names (*e.g.*, out) rejected by MySQL, bad foreign key declarations or other incompatibilities detected when the projects were forced to use the InnoDB back-end instead of MyISAM, or even some PostgreSQL table definitions including a MySQL specific syntax that were clearly never tested. A particular pitfall of PostgreSQL is that by default syntax errors in statements from an SQL script are ignored and the interpreter simply jumps to the next statement. When installing a project, the flow of warnings often hides these errors. Turning off this feature requires modifying the script, as no command option disables it. More than a dozen PostgreSQL projects contained this kind of issues, which resulted in missing tables or ignored constraint declarations.

Project	Total		MySQL		PgSQL		Both		Tables		Atts/table	
category	nb	%	nb	%	nb	%	nb	%	avg	med	avg	med
CMS	83	16.2	71	18.4	1	3.3	11	11.5	36.6	23	6.6	6.7
System	48	9.4	26	6.7	1	3.3	21	21.9	25.2	9	10.9	7.1
Project	28	5.5	15	3.9	5	16.7	8	8.3	25.4	19	6.9	7.0
Blog	27	5.3	22	5.7	0	0.0	5	5.2	26.8	21	6.9	6.8
Market	22	4.3	21	5.4	0	0.0	1	1.0	53.0	28	7.6	7.2
Forum	19	3.7	17	4.4	0	0.0	2	2.1	23.1	19	8.3	8.6
Accounting	18	3.5	11	2.8	6	20.0	1	1.0	87.8	45	8.8	8.8
Game	16	3.1	16	4.1	0	0.0	0	0.0	26.4	22	6.6	6.9
Mail	16	3.1	8	2.1	1	3.3	7	7.3	10.1	6	5.4	5.0
IRC	13	2.5	6	1.6	1	3.3	6	6.3	14.3	15	6.8	5.8
Homepage	12	2.3	11	2.8	0	0.0	1	1.0	5.1	4	7.0	7.0
Healthcare	11	2.1	6	1.6	2	6.7	3	3.1	89.5	71	11.5	9.5
Phone	11	2.1	5	1.3	2	6.7	4	4.2	18.2	9	14.6	9.0
Address	10	2.0	10	2.6	0	0.0	0	0.0	7.7	7	7.7	7.9
Genealogy	10	2.0	8	2.1	1	3.3	1	1.0	16.4	12	8.4	8.6
Photo	10	2.0	9	2.3	0	0.0	1	1.0	20.2	16	7.1	7.3
Community	9	1.8	7	1.8	0	0.0	2	2.1	17.3	12	8.1	8.0
Music	9	1.8	8	2.1	1	3.3	0	0.0	16.7	8	5.0	6.0
P2P	9	1.8	8	2.1	0	0.0	1	1.0	11.9	7	7.0	8.0
Reference	9	1.8	8	2.1	0	0.0	1	1.0	15.8	16	11.7	8.0
Wiki	9	1.8	7	1.8	1	3.3	1	1.0	15.7	9	5.6	5.7
Calendar	8	1.6	7	1.8	1	3.3	0	0.0	11.1	8	6.1	6.8
Advert	7	1.4	7	1.8	0	0.0	0	0.0	4.0	2	9.0	8.4
Search	6	1.2	6	1.6	0	0.0	0	0.0	18.0	20	6.0	6.0
Student	6	1.2	6	1.6	0	0.0	0	0.0	35.5	28	6.5	6.7
Teaching	6	1.2	3	0.8	1	3.3	2	2.1	13.5	5	4.9	5.3
Conference	5	1.0	4	1.0	1	3.3	0	0.0	73.8	32	6.8	6.2
FAQ	5	1.0	3	0.8	0	0.0	2	2.1	25.0	30	6.6	5.3
Library	5	1.0	4	1.0	1	3.3	0	0.0	63.8	72	7.2	7.3
Survey	5	1.0	3	0.8	0	0.0	2	2.1	25.0	18	6.4	6.4

TABLE I
MAIN CATEGORIES OF PROJECTS, WITH COUNTS, DATABASE SUPPORT AND SIZES

Project	Total		MySQL		PgSQL		Both		Tables		Atts/table	
technology	nb	%	nb	%	nb	%	nb	%	avg	med	avg	med
PHP	399	77.9	335	86.8	8	26.7	56	58.3	29.3	16	7.4	7.2
C	38	7.4	12	3.1	5	16.7	21	21.9	21.3	9	11.5	8.3
Java	22	4.3	8	2.1	6	20.0	8	8.3	67.5	23	9.3	8.2
Perl	21	4.1	10	2.6	5	16.7	6	6.3	44.0	29	6.7	6.7
SQL	8	1.6	6	1.6	1	3.3	1	1.0	27.3	11	4.9	5.0
C++	7	1.4	5	1.3	1	3.3	1	1.0	11.4	6	15.3	6.0
Python	7	1.4	4	1.0	2	6.7	1	1.0	42.9	17	6.5	6.2
Ruby	7	1.4	4	1.0	2	6.7	1	1.0	49.5	16	7.4	6.7

TABLE II
MAIN TECHNOLOGIES OF PROJECTS, WITH COUNTS, DATABASE SUPPORT AND SIZES

B. Overview of projects

We have studied the relational schemas of 512 (see appendix for the full list) open-source projects based on databases: 482 of these run with MySQL, 126 with PostgreSQL, including 96 on both. A project supporting PostgreSQL is very likely to support also MySQL (76%), although the reverse is not true (only 19%) *(very sure)*, outlining the relative popularity of these tools. Only 30 projects are PostgreSQL specific. Although there is no deliberate bias in the selection process described in the previous section, where we aimed at completeness, some implicit bias remain nevertheless: for instance, as we can speak mostly English and French, we found mostly international projects advertised in these tongues; Table I shows main project categories, from the personal mundane (game, homepage) to the professional serious (health-care, accounting, system). Table II shows the same for project technologies. Projects in rare categories or using rare technologies do not appear in these cut-off tables. The result is heavily slanted towards PHP web applications (77%), which seems to reflect the current trend of open-source programming as far as the number of projects is concerned, without indication of popularity or quality. The ratio of PHP projects increases from PostgreSQL only support (26%) to both database support (58%) *(very sure)* to MySQL only support (86%) *(very sure)*: PHP users tend to choose specifically MySQL, possibly because of traditional LAMP *(Linux, Apache, MySQL, PHP)* setups advertised with PHP programming. For instance, a search on the Amazon website in January 2012 returns 18 times more results with *PHP MySQL* compared to *PHP PostgreSQL*.

The survey covers 18993 tables (MySQL 13494, Post-

Advice	*Lvl.*	*Cat.*	*Sev.*	MySQL				PostgreSQL			
				Proj	*%*	*Adv*	*%*	*Proj*	*%*	*Adv*	*%*
Schema without any FK	sch.	design	error	425	88	425	88	70	55	70	55
Tables without PK nor Unique	table	design	error	262	54	1521	11	76	60	1010	18
FK type mismatch	table	consist.	error	2	0	17	0	10	7	153	2
Backend engine inconsistency	sch.	version	error	30	6	30	6	0	0	0	0
FK length mismatch	table	consist.	error	4	0	6	0	2	1	10	0
Integer PK but no other key	table	design	warn	437	90	7470	55	106	84	2509	45
Homonymous heterogeneous attributes	att.	style	warn	296	61	2294	2	76	60	573	1
Unsafe backend engine used in schema	sch.	version	warn	433	89	433	89	0	0	0	0
Attribute count per table over 40	table	design	warn	98	20	220	1	25	19	91	1
Isolated Tables	table	design	warn	30	6	979	7	40	31	1300	23
Tables without PK but with Unique	table	design	warn	117	24	405	3	15	11	40	0
Unique nullable attributes	att.	design	warn	73	15	261	0	23	18	172	0
Nullable attribute rate over 80%	sch.	design	warn	34	7	34	7	25	19	25	19
Redundant indexes	table	system	warn	0	0	0	0	23	18	196	3
Attribute name length too short	att.	style	warn	27	5	91	0	16	12	51	0
Large PK referenced by a FK	table	design	warn	10	2	118	0	19	15	216	3
Table name length too short	table	style	warn	16	3	23	0	7	5	17	0
Composite Foreign Key	table	design	warn	5	1	19	0	8	6	26	0
FK not referencing a PK	table	design	warn	2	0	16	0	7	5	23	0
Redundant FK	table	system	warn	1	0	1	0	2	1	6	0
Non-integer Primary Key	table	design	note	268	55	2261	16	81	64	1729	31
MySQL is used	base	version	note	482	100	482	100	0	0	0	0
Attribute count per table over 20	table	design	note	230	47	684	5	60	47	421	7
Tables with Composite PK	table	design	note	196	40	1781	13	63	50	703	12
Attribute name length quite short	att.	style	note	201	41	748	0	49	38	244	0
Attribute named after its table	att.	style	note	139	28	3114	2	42	33	5033	9
Table without index	table	system	note	0	0	0	0	60	47	719	13
Nullable attribute rate in 50-80%	sch.	design	note	76	15	76	15	33	26	33	26
Table name length quite short	table	style	note	70	14	102	0	28	22	52	0
Table with a single attribute	table	design	note	74	15	419	3	26	20	91	1
Mixed attribute name styles	table	style	note	115	23	1007	7	1	0	37	0
Mixed table name styles	sch.	style	note	51	10	261	54	8	6	22	17
Attribute name length short	att.	style	info	326	67	2911	2	81	64	1047	2
Unsafe backend engine used on table	table	version	info	433	89	10423	77	0	0	0	0
Nullable attribute rate in 20-50%	sch.	design	info	137	28	137	28	41	32	41	32
Table name length short	table	style	info	136	28	258	1	38	30	81	1

TABLE III
LIST OF RAISED ADVICES AND DETAILED COUNTS ABOUT THE 512 PROJECTS

greSQL 5499) containing 166906 attributes (MySQL 114561, PostgreSQL 52345). The project sizes average at 31.2 tables, median 16 (from 1 to 607), with 2 to 10979 attributes. MySQL projects average at 28 tables, median 15 (from 1 to 466), with 238 attributes (from 2 to 9725), while PostgreSQL projects average 44 tables, median 18 (from 1 to 607), with 415 attributes (from 5 to 10979 attributes). The largest MySQL project is OSCARMCMASTER, and the largest PostgreSQL project is ADEMPIERE. Detailed table counts raise from projects with MySQL only support (average 26.4, median 15), to both databases (average 34.0, median 17) or PostgreSQL only (average 75.5, median 30.5). MySQL-only projects are smaller than other projects *(marginally sure)*: more ambitious projects seem to use feature-full but maybe less easy to administrate PostgreSQL. However obvious this assertion would seem, the statistical validation is weak because of the small number of projects with PostgreSQL. MySQL projects that use the InnoDB back-end are much larger that their MyISAM counterpart *(very sure)* and are comparable to projects based on PostgreSQL, with 53 tables on average. The number of attributes per table is comparable although smaller for MySQL (average 8.5 – median 7.0) with respect to PostgreSQL (average 9.5 – median 6.0).

The per-category tables and attributes-per-table counts shows that *accounting*, *health-care* and *market* projects seem more ambitious than other categories *(marginally sure)*. The per-technology analysis counts suggests that *Perl*, *Python* and *Java* projects are larger than those based on other technologies *(marginally sure)*.

These projects are mostly recent, at least according to their status at an arbitrary common reference date chosen as March 31, 2009: 310 (60%) were updated in the last year, including 179 (34%) in the last six months, and the others are either obsolete or stable. The rate of recently updated projects raises from MySQL-only projects (55%) to projects with both support (73%) *(very sure)* or with PostgreSQL support at (76%) *(very sure)*, but there is no significant difference on the recent maintenance figures between projects that are PostgreSQL-only and projects with both databases support.

New data about the status of projects were collected on January 9, 2012. We could not find 69 projects in this new survey (61 MySQL-only, 1 PostgreSQL-only and 7 with both support). Moreover, 153 projects are stale, that is not

updated between the 2009 and 2012 data (128 MySQL-only, 6 PostgreSQL-only and 19 with both support). Nearly half of the MySQL projects are stale or lost, while it is only one quarter of the PostgreSQL projects. MySQL-only projects are more often lost or stale than others in 2012 *(very sure)*, and it is still true for MySQL projects compared to PostgreSQL-only projects *(rather sure)*. More generally, on these new data, MySQL-only projects are less maintained than others *(very sure)*, and it is still true compared to projects with both support *(very sure)* and compared to projects with PostgreSQL-only support *(rather sure)*. There are about six months (180 days) between the median update date of MySQL-only projects and PostgreSQL-only projects. Even if we ignore lost and stale projects to focus on projects that were indeed updated in the 2012 data, PostgreSQL-only projects were more recently updated than others *(rather sure)*. Yet again, there is no significant update status difference between projects with PostgreSQL support and projects that support both databases on the 2012 data. To conclude, the maintenance of PostgreSQL projects seems more intense: projects that include PostgreSQL support were updated more recently both in 2009 and in 2012.

IV. Survey results

We now analyze the open-source projects of our survey by commenting actual results on MySQL and PostgreSQL, before comparing them. Table III summarizes the advices raised for MySQL and PostgreSQL applications. The first four columns give the advice title, level, category and severity. Then four columns for each database list the results. The first two columns hold the number of projects (*i.e.* schema) tagged and the overall rate. The last two columns give the actual number of advices and rate, which varies depending on the level. A per-project aggregate is also available online [71].

A. Primary keys

A majority of MySQL projects (262 – 54%) have at least one table without neither a primary key nor a unique constraint, and this is even worse with PostgreSQL projects (76 – 60%). The certainty of the observation *(rather sure)* on MySQL-only vs PostgreSQL-only is low because of the small number of projects using the later. As 11% of all MySQL tables and 18% of all PostgreSQL tables do not have any key, the view of relations as sets is hindered as tuples are not identified, and data may be replicated without noticing.

A further analysis gives some more insight. For MySQL, 41% of tables without key do have some `KEY` option for indexes, but without the `UNIQUE` or `PRIMARY` keyword that makes it a key. Having `KEY` not always declaring a key was clearly a bad design choice. A little 5% of tables without key have an *auto increment* attribute, which suggest uniqueness in practice, but is not enforced. Also, the missing key declaration often seems to be composite. Some tables without key declarations are intended as one tuple only, say to check for the version of the schema or configuration of the application. Similarly, 28% of PostgreSQL tables without key have an index declared. Moreover, 22% have a `SERIAL` (auto incremented) attribute: Many designers seem to assume wrongly that `SERIAL` implies a key. A comment found in the `SQLGREY` project source suggests that some keys are not declared because of MySQL key size limits.

A simple integer primary key is provided on 61% of tables, with a significantly decreasing rate from MySQL-only (65%) to both database support (62%) *(rather sure)* down to PostgreSQL-only support (39%) *(very sure)*. If these primary keys were non-semantic numbers to identify tuples, one would expect at least one other key declared on each table to identify the underlying semantic key. However it is not the case: most (85%) of these tables do not have any other key. When a non simple primary key is available, it is either based on another type or a composite key. The composite keys are hardly referenced, but as the foreign keys are rarely declared one cannot be sure, as shown in the next section.

B. Referential integrity

Foreign keys are important for ensuring data consistency in relational databases. They are supported by PostgreSQL, and by MySQL but with some back-end engines only. In particular, the default MyISAM back-end does not support foreign keys, and this feature was deemed noxious in previous documentations: Version *3.23* includes a *Reasons NOT to Use Foreign Keys constraints* Section arguing that they are only useful to display diagrams, hard to implement and terrible for performance. Foreign key constraints are introduced with the InnoDB engine starting with *MySQL 3.23.44* in January 2001. Although the constraints are ignored by the default MyISAM engine, the syntax is parsed, and triggers the creation of indexes. Version *5.1* documentation has a *Foreign Keys* Section praising the feature, as it *offers benefits*, although it slows down the application. Caveats describe the inconsistencies that may result from *not* using transactions and referential integrity. From a pedagogical perspective, this is a progress.

Foreign key constraints have long been a missing or avoided feature in MySQL and this seems to have retained momentum in many projects, as it is not supported by the default engine: few MySQL projects (57 – 11% of all projects, but 72% of those with InnoDB) use foreign key constraints. The foreign key usage rate is slightly higher (20%) when considering projects supporting both databases *(marginally sure)*.

Among MySQL projects, 403 (83%) use only the default MyISAM back-end engine, thus do not have any foreign key checks enabled. In the remainder, 49 (10%) use only InnoDB, and 30 (6%) use a combination of both. More projects (21 – 21%) rely on InnoDB among those supporting both MySQL and PostgreSQL *(marginally sure)*. A third of InnoDB projects (30 – 37%) are not consistent in their engine choice: 34% of tables use MyISAM among the 79 InnoDB projects. A legitimate reason for using MyISAM tables in an InnoDB project is that full-text indexes are only available with the former engine. However, this only applies to 11 tables in 6 projects, all other 1441 MyISAM tables in InnoDB projects are not justified by this. A project may decide to store transient data in an unsafe engine (*e.g.*, memory) for performance reason. However, this case is rare, as it represents only 15 tables in 8 projects. About 26% of tables use MyISAM as a default implicit choice in

InnoDB projects, similar to 28% when considering all MySQL projects. Some engine inconsistencies seems due to forgotten declarations falling back to the default MyISAM engine.

We have forced the InnoDB back-end engine for all MySQL projects: 22 additional projects declare 92 new foreign key constraints previously ignored. These new foreign keys are very partial, targeting only some tables. They allow to uncover about two dozen issues, either because the foreign key declaration were failing (say from type errors detected by MySQL) or thanks to analyses from our tool. Additional checks based on foreign keys cannot be raised on schemas that do not declare any of them. Thus *isolated tables* warnings must be compared to the number of projects that do use referential constraints: 30 – 52% of these seem to have forgotten at least some foreign keys, and it is actually the case by checking some of these projects manually.

The foreign key usage is better with PostgreSQL projects, although it is still a minority (56 projects – 44%). This rate is close to the foreign key usage of MySQL projects when considering InnoDB projects only. It gives a better opportunity for additional advices to be checked. The foreign key usage rate raises significantly to 74% when considering PostgreSQL-only projects vs dual support projects *(very sure)*.

On the very few projects with partial foreign key declarations, several of these declaration reveal latent bugs, including type mismatch, typically `CHAR` targeting a `VARCHAR` or vice versa, or different integers, and type length mismatch, usually non matching `VARCHAR` sizes. We found 23 such bugs out of the small 1979 declared MySQL attribute constraints, and 163 among the 4424 PostgreSQL constraints. The rate is greater for PostgreSQL, possibly helped by the use of `SERIAL` which may be considered as a primary key by developers without being declared as such. There are also 153 important warnings related to foreign keys raised for MySQL, and 265 for PostgreSQL. If this error ratio is extrapolated to the number of tables, hundreds of additional latent bugs could be detected using the missing referential constraints.

C. Miscellaneous issues

More issues were found about style, attribute constraints and by comparing projects with dual database support.

There is 13669 noticeable style issues raised from our analyses (7640 for MySQL, 6029 for PostgreSQL), relating to table or attribute names, including a number of one-letter attribute names or two-letters table names. The *id* attribute name is used in the SLASH project with up to 6 different types, mixing various integers and fixed or variable length text types. In PHPETITION, a *date* attribute has types `DATE`, `DATETIME` or `VARCHAR`. 81% of MySQL projects and 78% of PostgreSQL have such style issues.

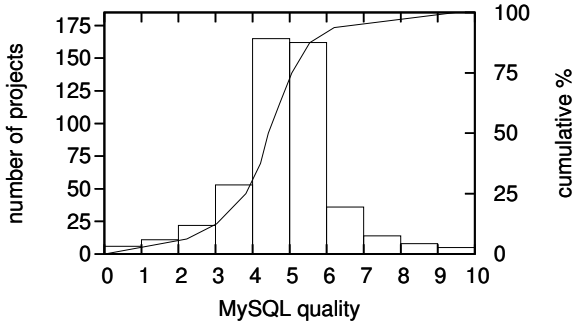
Many projects do not bother with `NOT NULL` attribute declarations: 110 MySQL projects (22%) and 58 PostgreSQL projects (46%) have over half of their attributes null-able. This does not reflect the overall use of constraints: for MySQL, the average number of key-related constraints per table is 1.07 (from BOARDPLUS 0.00 to JWHOISSERVER 3.57), while for PostgreSQL it is 1.24 (from ANDROMEDA 0.00 to ADEMPIERE 4.25). Project ANDROMEDA is astonishing: there is not a single constraint declared (no primary key, no foreign key, no unique, no not null) on the 180 tables, although there are a number of non-unique indexes and of sequences.

It is interesting to compare the schemas of the 96 projects available with both databases. This dual support must not be taken at face value: PostgreSQL support is often an afterthought and is not necessarily functional, including project such as ELGG, TAGADASH, QUICKTEAM or TIKIWIKI where some PostgreSQL table declarations use an incompatible MySQL syntax; 38 (39%) projects have missing tables or attributes between the MySQL and PostgreSQL versions: 398 tables and 191 individual attributes are missing or misspelled one side or another. Among the missing tables, 73 look like some kind of sequence, and thus might be possibly legitimate, although why the *auto increment* feature was not satisfactory is unclear. At the minimum, the functionalities are not the same between the MySQL and PostgreSQL versions of these projects.

D. Overall quality

We have computed a synthetic project quality evaluation ranging from 10 (good) to 0 (bad) by removing points based on advice severity (error, warning, notice), level (schema, table, attribute) and project size. The MySQL projects quality average is 4.4 ± 1.4 (from 9.5 JWHOISSERVER to 0.0 MANTIS), significantly lower than PostgreSQL 5.4 ± 1.8 (from 9.4 COMICS to 0.0 NURPAWIKI) *(very sure)*. This does not come as a surprise: most MySQL projects choose the default data-unsafe MyISAM engine, hence incur a penalty. Also, the multiplicity of MySQL back-ends allows the user to mix them unintentionally, what is not possible with PostgreSQL. When all MySQL-specific advices are removed, the quality measure is about the same for both databases. However, as PostgreSQL schemas provide more information about referential integrity constraints, they are also penalized as more advices can be raised based on the provided additional information. For projects which support both databases, the grade's correlation is significant and positive (0.55), which is logical as the same style warnings are triggered on both sides.

Table IV shows the projects per quality decile. The PostgreSQL-only project quality is more spread than MySQL projects *(very sure)*. Table V compares the quality of projects according to size, with small up to 9, medium up to 29, and large otherwise. The quality is quite evenly distributed among sizes, which suggests that our effort to devise a size-neutral grading succeeded. Table VI compares quality based on the project categories. The number of projects in each category is too small to draw deep conclusions. Table VII addresses the technology used in the project: Java and Python lead while C, PHP and Ruby are near bottom. PHP projects take less care of their relational design *(rather sure)*, but this may be explained by the fact that MySQL is used more often in these projects, and that an unsafe engine is selected more often *(very sure)*. Yet again, the very small count of projects with some of the technologies do not allow to draw deep conclusion about them. Finally, Table VIII and Table IX show that quality evaluation does not change much depending whether projects are updated more often.



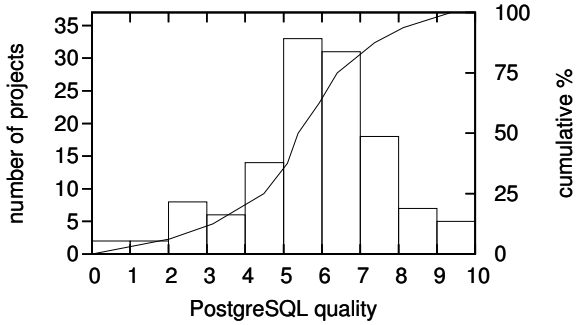


TABLE IV
QUALITY PER DECILE

MySQL projects

Size	nb	%	avg	σ	min	med	max
small	181	38	4.7 ± 1.4		0.0	4.5	9.5
medium	164	34	4.2 ± 1.3		0.0	4.3	8.7
large	137	28	4.3 ± 1.4		0.0	4.4	8.2

PostgreSQL projects

Size	nb	%	avg	σ	min	med	max
small	44	35	5.3 ± 2.0		0.0	5.3	9.4
medium	37	29	5.5 ± 1.5		2.0	5.3	9.3
large	45	36	5.3 ± 2.0		0.0	5.7	8.1

TABLE V
QUALITY PER SIZE

MySQL projects

Category	nb	%	avg	σ	min	med	max
irc	12	2	5.1 ± 1.3		2.0	5.4	7.0
mail	15	3	4.4 ± 1.7		1.7	4.7	8.4
project	23	5	4.3 ± 1.4		0.0	4.6	6.2
system	47	10	4.5 ± 1.4		0.0	4.5	9.5
game	16	3	4.4 ± 2.0		0.9	4.5	9.1
blog	27	6	4.4 ± 0.9		2.5	4.5	7.2
forum	19	4	4.3 ± 0.9		2.4	4.4	5.7
cms	82	17	4.2 ± 1.1		0.0	4.3	8.3
homepage	12	2	4.1 ± 0.9		3.0	4.1	5.9
market	22	5	4.0 ± 1.4		1.8	4.0	8.2
accounting	12	2	4.4 ± 1.9		1.9	3.6	7.5

PostgreSQL projects

Category	nb	%	avg	σ	min	med	max
teaching	3	2	7.9 ± 2.2		5.3	8.9	9.4
blog	5	4	6.6 ± 1.1		5.3	6.4	8.2
accounting	7	6	5.9 ± 2.0		2.0	6.4	7.8
cms	12	10	6.1 ± 1.3		4.0	5.9	8.1
irc	7	6	5.4 ± 1.7		2.0	5.6	7.4
phone	6	5	5.2 ± 1.5		3.1	5.3	7.4
project	13	10	5.4 ± 1.6		2.2	5.2	9.3
system	22	17	5.0 ± 2.1		1.6	5.1	9.0
mail	8	6	4.9 ± 1.6		3.0	4.8	7.5
healthcare	5	4	3.2 ± 2.7		0.0	3.3	6.6

TABLE VI
QUALITY PER PROJECT MAIN CATEGORIES

MySQL projects

Techno.	nb	%	avg	σ	min	med	max
python	5	1	5.9 ± 2.0		3.7	6.2	8.2
sql	7	1	4.0 ± 2.5		0.0	5.3	5.9
java	16	3	4.8 ± 2.8		0.0	5.2	9.5
c++	6	1	4.8 ± 1.2		3.3	4.5	7.0
c	33	7	4.6 ± 1.4		2.0	4.4	8.4
php	391	81	4.4 ± 1.2		0.0	4.4	9.1
perl	16	3	3.9 ± 2.1		0.0	4.3	8.7
ruby	5	1	4.5 ± 0.9		3.7	4.2	5.6

PostgreSQL projects

Techno.	nb	%	avg	σ	min	med	max
python	3	2	7.0 ± 0.6		6.6	6.8	7.7
java	14	11	6.1 ± 2.4		0.0	6.8	9.3
c++	2	2	6.7 ± 1.0		6.0	6.7	7.4
perl	11	9	6.0 ± 1.9		2.0	6.1	8.9
sql	2	2	5.8 ± 5.1		2.2	5.8	9.4
php	64	51	5.2 ± 1.6		0.0	5.4	8.2
ruby	3	2	5.1 ± 1.2		4.0	5.0	6.3
c	26	21	4.8 ± 1.9		1.6	5.0	9.0

TABLE VII
QUALITY PER PROJECT MAIN TECHNOLOGIES

MySQL projects

Date	nb	%	avg	σ	min	med	max
recent	162	34	4.3 ± 1.3		0.0	4.4	8.6
older	320	66	4.4 ± 1.4		0.0	4.4	9.5

PostgreSQL projects

Date	nb	%	avg	σ	min	med	max
recent	59	47	5.3 ± 1.6		0.0	5.3	9.3
older	67	53	5.4 ± 2.0		0.0	5.6	9.4

TABLE VIII
QUALITY PER PROJECT UPDATE IN MARCH 2009

MySQL projects

Date	nb	%	avg	σ	min	med	max
recent	112	23	4.3 ± 1.3		0.0	4.5	7.8
older	155	32	4.4 ± 1.5		0.0	4.5	9.5
stale	147	30	4.5 ± 1.4		0.9	4.4	9.1
lost	68	14	4.2 ± 1.0		0.0	4.2	6.3

PostgreSQL projects

Date	nb	%	avg	σ	min	med	max
recent	41	33	5.7 ± 1.4		0.0	5.6	7.7
older	52	41	5.2 ± 1.9		0.0	5.3	9.3
stale	25	20	5.1 ± 2.2		0.7	5.3	9.4
lost	8	6	5.5 ± 2.3		2.0	5.5	9.0

TABLE IX
QUALITY PER PROJECT UPDATE IN JANUARY 2012

V. Conclusion

This is the first survey on the quality of relational schemas in open-source software. The overall quality results are worse than envisioned at the beginning of the study. Although we did not expect a lot of perfect projects, having so few key declarations and referential integrity constraints came as a surprise. We must acknowledge that our assumption that data are precious, and that the database should help preserve its consistency by enforcing integrity constraints and implementing transactions, is not shared by most open-source projects, especially when based on MySQL and PHP. This is illustrated by bug report 15441 [79] about missing keys on tables in MEDIAWIKI, the software behind Wikipedia: it had no effect on the software after more than three years, although it triggered some discussions at the beginning of 2012.

We can only speculate about the actual reasons that explain the poor quality of the surveyed schemas in open-source projects. One way to investigate further these issues would be to collect data about and from the people who designed the relational schemas of these projects. For instance, if MySQL or PHP users are found less savvy about software development, that could account for a lower quality and maintenance of the corresponding projects. Some interesting questions could be investigated: What are their educational and professional background? Did they receive any formal education about computer programming in general? About relational database design in particular? Do they consider database design as an important issue? How are they perceiving the actual quality of their schemas, and the quality of their software? When did they started database design? For MySQL, what database engines do they use? Did the initial policy of discouraging foreign key usage influence them? We attempted to conduct such a survey by contacting some people by e-mail and encouraging them to fill a web form online. The return ratio of this survey attempt was *null*. This establishes the fact that schema designers in open-source software do not wish to answer such questions, with a very high degree of accuracy.

Another relevant question is whether our results would be different if we studied closed-source projects developed by payed professionals, possibly using non open-source database technologies from Oracle or Microsoft. However, accessing such data at a level compatible with statistical validation seems very difficult. If we were to believe some of our experience, the results could end up being quite similar, especially when considering PHP/MySQL projects.

It is interesting to note that the first author contributed both to the best PostgreSQL project (COMICS), and to one of the worst MySQL project (SLXBBL), which is *Salix* executed on its own schema. This deserves an explanation: COMICS is a small database used for teaching SQL. The normalized schema emphasizes clarity and cleanliness with a pedagogic goal in mind. Even so, the two raised warnings deserve to be fixed, although one would require an additional attribute. SLXBBL tables generate a lot of errors, because they are views materialized for performance issues. Also, they rely on MyISAM because some SQL create table statements must be compatible with both MySQL and PostgreSQL to ease the tool portability. Nevertheless, the comparison of schemas allowed to find one bug: an attribute had a different name, possibly because of a bad copy-paste.

Acknowledgement

We are indebted to Pierre Jouvelot for helping with the title and proof reading. We also thank the anonymous reviewers for their helpful remarks that we tried to address for the better of the paper.

References

[1] F. Coelho, A. Aillos, S. Pilot, and S. Valeev, "A Field Analysis of Relational Database Schemas in Open-source Software," in *DBKDA: 3rd Int. Conf. on Advances in Databases, Knowledge, and Data Applications*, IARIA, Ed., no. ISBN:978-1-61208-002-4, St Marteen, The Netherlands Antilles, Jan. 2011, pp. 9–15.

[2] J. M. Gonzales-Barahona, P. Heras Quiros, and T. Bollinger, "A brief history of free software and open source," *IEEE Software*, pp. 32–33, Jan. 1999.

[3] R. Stallman, "GNU Project announcement," http://www.gnu.org/gnu/initial-announcement.html *(2012-01-06)*, Sep. 1983.

[4] ——, "FSF: Free Software Foundation," Oct. 1985, www.fsf.org, *(2012-01-06)*.

[5] A. Deshpande and D. Riehle, "The Total Growth of Open Source," in *4th Conf. on Open Source Systems (OSS)*. Springer Verlag, 2008, pp. 197–209.

[6] L. F. Wurster, "As Number of Business Processes Using Open-Source Software Increases, Companies Must Adopt and Enforce an OSS Policy," Gartner Inc, Sep. 2008, iD Number: G00160997.

[7] D. C. Plummer, B. Gammage, K. Harris-Ferrante, and J. Lopez, "Predicts 2010: Revised Expectations for IT Demand, Supply and Oversight," Gartner, Inc, Dec. 2009, iD Number: G00173560.

[8] "Open Source Licences," http://opensource.org *(2012-01-06)*, Feb. 1998.

[9] K. Crowston, H. Annabi, and J. Howison, "Defining open source software project success," in *24th Int. Conf. on Information Systems (ICIS)*, 2003, pp. 327–340.

[10] S. Görling, "A critical approach to open source software," http://flosshub.org/196 *(2012-01-06)*, 2003.

[11] C. Gacek, T. Lawrie, and B. Arief, "The many meanings of open source," *IEEE Software*, vol. 21, pp. 34–40, 2004.

[12] E. von Hippel, B. Mako Hill, and K. Lakhani, "Free and opensource software research community," http://opensource.mit.edu, now offline, Nov. 2001.

[13] A. Hars, "Working for free? motivations for participating in open-source projects," *Int. J. of Electronic Commerce*, vol. 6, pp. 25–39, 2002, also IEEE 34th Hawaii Int. Conf. on System Sciences 2001.

[14] G. Hertel, S. Niedner, and S. Herrmann, "Motivation of software developers in open source projects: An internet-based survey of contributors to the linux kernel," *Research Policy*, vol. 32, pp. 1159–1177, 2003.

[15] I. horn Hann, J. Roberts, S. Slaughter, and R. Fielding, "An empirical analysis of economic returns to open source participation (unpublished working paper)," 2004.

[16] A. Bonaccorsi and C. Rossi, "Altruistic individuals, selfish firms? the structure of motivation in open source software," Santa Anna School of Advanced Studies. Institute for Informatics and Telematics, Tech. Rep., Jan. 2004, Fist Monday, http://firstmonday.org/ *(2012-01-06)*.

[17] K. J. Stewart and S. Gosain, "The impacts of ideology on effectiveness in open source software development teams (working paper)," *MIS Quarterly*, vol. 30, pp. 291–314, 2005.

[18] J. E. Cook, "Open source development: An arthurian legend. making sense of the bazaar," in *Proceedings of the 1st Workshop on Open Source Software*, 2001.

[19] M. S. Elliott and W. Scacchi, "Mobilization of software developers: The free software movement," 2006.

[20] ——, "Free software: A case study of software development in a virtual organizational culture," in a Virtual Organizational Culture, Working Paper, Institute for Software Research, Tech. Rep., 2003.

[21] M. S. Elliott, "Free software developers as an occupational community: Resolving conflicts and fostering," in *Collaboration, Proc. ACM Int. Conf. Supporting Group Work*, 2003, pp. 21–30.

[22] K. Healy and A. Schussman, "The ecology of open-source software development," Dept of Sociology, Univ. of Arizona, Tech. Rep., 2003.

[23] K. Crowston and H. Annabi, "Effective work practices for software engineering: Free/libre open source software development," in *in Proc. of WISER*. ACM Press, 2004, pp. 18–26.

[24] W. Seidel and C. Niedermeier, "Open source software: Leveraging software quality in the industrial context," OSSIE, 2003.

[25] W. Scacchi, J. Feller, B. Fitzgerald, S. Hissam, and K. Lakhani, "Understanding Free/Open Source Software Development Processes," *Software Process: Improvement and Practice*, vol. 11, no. 2, pp. 95–105, May 2006.

[26] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter, "An empirical study of global software development: Distance and speed," in *In 23nd Int. Conf. on Software Engineering*. IEEE Computer Society, 2001, pp. 81–90.

[27] B. J. Dempsey, D. Weiss, P. Jones, and J. Greenberg, "A quantitative profile of a community of open source linux developers," University of North Carolina at Chapel Hill, Tech. Rep., 1999.

[28] R. A. Ghosh, R. Glott, B. Krieger, and G. Robles, "Free/libre and open source software: Survey and study, floss, part 4: Survey of developers," Int. Institute of Infonomics, University of Maastricht, The Netherlands, Tech. Rep., Jun. 2002.

[29] D. M. Nichols and M. B. Twidale, "The usability of open source software," *First Monday*, vol. 8, 2003.

[30] Eclipse Foundation, "The open source developer report, 2010 eclipse community survey," Tech. Rep., Jun. 2010.

[31] J. Lerner and J. Tirole, "The economics of technology sharing: open source and beyond. working paper 10956. retrieved jun 7, 2005 http://www.nber.org/papers/w10956," *J. of Economic Perspectives*, vol. 19, pp. 99–120, 2004.

[32] K. M. Schmidt and M. Schnitzer, "Public subsidies for open source? some economic policy," 2002, cEPR Discussion Paper 3793.

[33] Netcraft Ltd, "Web Server Survey," http://news.netcraft.com/ *(2012-01-06)*, 2012, running since 1995.

[34] A. Mockus, R. T. Fielding, and J. Herbsleb, "Two case studies of open source software development: Apache and mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, pp. 309–346, 2002.

[35] K. R. Lakhani, "How open source software works: "free" user-to-user assistance," *Research Policy*, pp. 923–943, 2000.

[36] B. Mishra, A. Prasad, and S. Raghunathan, "Quality and Profits Under Open Source Versus Closed Source," in *ICIS*, no. 32, 2002.

[37] I. Stamelos, L. Angelis, A. Oikonomou, and G. L. Bleris, "Code quality analysis in open-source software development," *Information Systems J., 2nd Special Issue on Open-Source*, vol. 12, no. 1, pp. 43–60, Feb. 2002, blackwell Science.

[38] E. Capra, C. Francalanci, and F. Merlo, "En Empirical Study on the Relationship among Software Design Quality, Development Effort and Governance in Open Source Projects," *IEEE Software Engineering*, vol. 34, no. 6, pp. 765–782, nov-dec 2008.

[39] R. Gobeille, "The FOSSology Project," in *Working Conf. on Mining Software Repositories*, no. 5, Leipzig, Germany, May 2008.

[40] G. Concas, M. Marchesi, A. Murgia, R. Tonelli, and I. Turnu, "On the distribution of bugs in the eclipse system," *IEEE Transactions on Software Engineering*, vol. 99, no. PrePrints, 2011.

[41] Coverty, "Coverty scan open source report," Coverty, White Paper, 2009.

[42] Veracode, Inc, "State of security report," White paper, Mar. 2010.

[43] C. Graham, D. Sommer, and B. Sood, "Market Share: Relational Database Management Systems by Operating System, Worldwide, 2006," Gartner, Inc, Jun. 2007, iD Number: G00149469.

[44] D. Litchfield, "The Database Exposure Survey 2007," NGSSoftware Insight Security Research (NISR), Nov. 2007.

[45] E. F. Codd, "A relational model for large shared databanks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970.

[46] ISO/IEC, "Information technology - database languages - SQL," 2003, standard 9075.

[47] ISO/IEC, Ed., *9075-11:2003: Information and Definition Schemas (SQL/Schemata)*. ISO/IEC, 2003.

[48] W. C. Burkett, "Database Schema Design Quality Principles," http://www.intergate.com/~wcb/DbSchemaQuality.pdf, *(2012-01-08)*, Dec. 1997.

[49] O. Herden, "Measuring Quality of Database Schemas by Reviewing – Concept, Criteria and Tool," in *5th Int. ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering (QAOOSE 2001)*, Budapest, Hungary, Jun. 2001.

[50] J. Lemaitre and J.-L. Hainaut, "Transformation-based Framework for the Evaluation and Improvement of Database Schemas," in *Int. Conf. on Advanced Information Systems Engineering (CAiSE)*, Hammamet, Tunisia, Jun. 2010.

[51] ——, "Quality Evaluation and Improvement Framework for Database Schemas Using Defect Taxonomies," in *Int. Conf. on Advanced Information Systems Engineering (CAiSE)*, London, United Kingdom, Jun. 2011.

[52] T. J. MacCabe, "A Complexity Measure," *IEEE Software Engineering*, vol. SE-2, no. 4, pp. 308–320, Dec. 1976.

[53] M. H. Halstead, *Elements of Software Science*. New York, USA: Elsevier, 1977, no. ISBN:0444002057.

[54] H. F. Li and W. K. Cheung, "An empirical study of software metrics," *IEEE Transactions on Software Engineering*, 1987.

[55] M. Piattini, M. Genero, C. Calero, and G. Alarcos, "Data model metrics," in *In Handbook of Software Engineering and Knowledge Engineering: Emerging Technologies, World Scientific*, 2002.

[56] M. Genero, "A survey of Metrics for UML Class Diagrams," *J. of Object Technology*, vol. 4, pp. 59–92, Nov. 2005.

[57] H. M. Sneed and O. Foshag, "Measuring legacy database structures," in *European Software Measurement Conf. (FESMA'98)*, Hooft and Peeters, Eds., 1998.

[58] M. Piattini, C. Calero, and M. Genero, "Table Oriented Metrics for Relational Databases," *Software Quality J.*, vol. 9, no. 2, pp. 79–97, 2001.

[59] A. L. Baroni, C. Calero, F. Ruiz, and F. Brito e Abreu, "Formalizing object-relational structural metrics," in *Conf. of APSI, Lisbon*, no. 5, Nov. 2004.

[60] C. Calero, M. Piattini, and M. Genero, "Empirical validation of referencial integrity metrics," *Information and Software Technology*, vol. 43, no. 15, pp. 949–957, Dec. 2001.

[61] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler, "A Few Billion Lines of Code Later: Using Static Analysis to Find Bugs in the Real World," *Communication of the ACM*, vol. 53, no. 2, pp. 66–75, Feb. 2010.

[62] A. Cleve, J. Lemaitre, J.-L. Hainaut, C. Mouchet, and J. Henrard, "The role of implicit schema constructs in data quality," in *Workshop on Management of Uncertain Data (MUD)*, Auckland, New Zealand, Aug. 2008, pp. 33–40.

[63] A. Aillos, S. Pilot, S. Valeev, and F. Coelho, "Salix Babylonica: advices about database relational schemas," Software from http://coelho.net/salix/ *(2012-01-06)*, Aug. 2008, version 1.0.0 on 2012-01-27.

[64] F. Coelho, "PG-Advisor: proof of concept SQL script," Mailed to `pgsql-hackers`, Mar. 2004.

[65] J. Currier, "SchemaSpy: Graphical database schema metadata browser," Source Forge, Aug. 2005, *(2012-01-06)*.

[66] B. Schwartz and D. Nichter, "Maatkit," Google Code, 2007, see *duplicate-key-checker* and *schema-advisor*. Part of the Percona Toolkit as of 2012-01-06 (http://www.percona.com/software/percona-toolkit/).

[67] J. Berkus, "Ten ways to wreck your database," O'Reilly Webcast, Jul. 2009, *(2012-01-06)*.

[68] A. M. Boehm, M. Wetzka, A. Sickmann, and D. Seipel, "A Tool for Analyzing and Tuning Relational Database Applications: SQL Query Analyzer and Schema EnHancer (SQUASH)," in *Workshop über Grundlagen von Datenbanken*, Jun. 2006, pp. 45–49.

[69] G. Singh, "PostgreSQL Adviser," Software at http://git.postgresql.org/gitweb/pg_adviser.git *(2012-01-06)*, Jul. 2007.

[70] E. F. Codd, "Is Your DBMS Really Relational? Does Your DBMS Run By The Rules?" *ComputerWorld*, Oct. 1985.

[71] F. Coelho, "Database quality survey projects and results," Jan. 2012, detailed list of projects surveyed in *On the Quality of Relational Database Schemas in Open Source Software*, report A/478/CRI. [Online]. Available: http://www.coelho.net/salix/projects.html

[72] K. Pearson, "On the Criterion that a Given System of Deviations from the Probable in the Case of a Correlated System of Variables is such that it Can Reasonably Be Supposed to have Arisen from Random Sampling," *Philosophical magazine*, vol. 5, no. 50, pp. 157–175, Jul-Dec 1900, Taylor & Francis Ed, London.

[73] MySQL AB, "MySQL – Relational Database Management System," http://mysql.com/ *(2012-01-06)*, May 1995.

[74] PostgreSQL Global Development Group, "PostgreSQL – Object-Relational Database Management System," http://postgresql.org/ *(2012-01-06)*, Aug. 1996, based on the Postgres, which started in 1986.

[75] "Debian," http://debian.org/ *(2012-01-06)*, Aug. 1993.

[76] Canonical Ltd, "Ubuntu," http://ubuntu.com/ *(2012-01-06)*, Oct. 2004.

[77] SecurityFocus, "Security advisories," http://securityfocus.com/ *(2012-01-06)*, Jan. 1999.

[78] "Source Forge," http://sourceforge.net/ *(2012-01-06)*, 1999.

[79] F. Coelho, "MediaWiki bug 15441," https://bugzilla.wikimedia.org/show_bug.cgi?id=15441 *(2012-01-06)*, Sep. 2008.

APPENDIX
LIST OF ADVICES

1) **Schema without any FK** *schema design error*
Why use a relational database if data are not related at all? Well, that might happen...
2) **No attribute in table** *table design error*
There must be something in a table.
3) **Tables without PK nor Unique** *table design error*
All tuples must be uniquely defined to be consistant with the set theory. There is no unique subset of attribute which can be promoted as a PK.
4) **Nullable attribute rate over 80%** *schema design warning*
Warning: Most of the time, attributes should be NOT NULL. Too high a rate of nullable attribute may reveal that some fields are lacking a NOT NULL.
5) **Attribute count per table over 40** *table design warning*
Having so many attributes in the same table may reveal the need for additional relations.
6) **Composite Foreign Key** *table design warning*
As for primary keys, simple foreign keys are usually better design, and make updates easier.
7) **FK not referencing a PK** *table design warning*
A Foreign Key should rather reference a Primary Key.
8) **Integer PK but no other key** *table design warning*
A simple integer primary key suggests that some other key must exist in the table.
9) **Isolated Tables** *table design warning*
In a database design, tables are usually linked together.
10) **Large PK referenced by a FK** *table design warning*
Having large primary keys referenced by a foreign key may reveal data duplication, as the primary key is likely to contain relevant information.
11) **Tables without PK but with Unique** *table design warning*
All tables should have a primary key to be consistant with the set theory. A unique constraint may be promoted as the primary key.
12) **Attribute has a pseudo 'NULL' text default** *attribute design warning*
Possibly the NULL value was intended instead of the 'NULL' text.
13) **Unique nullable attributes** *attribute design warning*
A unique nullable attribute may be a bad design if NULL does not have a particular semantic.
14) **Nullable attribute rate in 50-80%** *schema design notice*
Notice: Most of the time, attributes should be NOT NULL. Too high a rate of nullable attribute may reveal that some fields are lacking a NOT NULL.
15) **Attribute count per table over 20** *table design notice*
Having many attributes in the same table may suggest the need for additional relations.
16) **Non-integer Primary Key** *table design notice*
Having integer primary keys without specific application semantics make updates easier.
17) **Table with a single attribute** *table design notice*
Possibly some more attributes are needed to have a semantic.
18) **Tables with Composite PK** *table design notice*
A simple primary key, without specific semantics, is usually a better design, and references through foreign keys are simpler.
19) **Nullable attribute rate in 20-50%** *schema design information*
Information: Most of the time, attributes should be NOT NULL. Too high a rate of nullable attribute may reveal that some fields are lacking a NOT NULL.
20) **FK length mismatch** *table consistency error*
A Foreign Key should have matching referencing and referenced type sizes.
21) **FK type mismatch** *table consistency error*
A Foreign Key should have matching referencing and referenced types.
22) **Destination table and FK in different schemas** *table consistency warning*
A constraint and its destination table are usually in the same schema.
23) **Source table and constraint in different schemas** *table consistency warning*
A constraint and its source table should be in the same schema.
24) **Table and index in different schemas** *table consistency warning*
An index and its table should be in the same schema.
25) **Tables linked but in different schemas** *table consistency notice*
Linked tables are usually in the same schema.
26) **Backend engine inconsistency** *schema version error*
Different backends are used in the same database. It may be legitimate to do so if a particular feature of one backend is needed, for instance full text indexes.
27) **Unsafe backend engine used in schema** *schema version warning*
An unsafe backend (e.g. MyISAM) used at least once lacks referential integrity, transaction support, and is not crash safe.
28) **MySQL is used** *database version notice*
MySQL lacks important features of the SQL standard, including missing set operators.
29) **Unsafe backend engine used on table** *table version information*
An unsafe backend (e.g. MyISAM) lacks referential integrity, transaction support, and is not crash safe.
30) **Schema name length too short** *schema style warning*
A schema name with less than 3 characters is really too short.
31) **Table name length too short** *table style warning*
A table name with less than 2 characters is really too short.
32) **Attribute name length too short** *attribute style warning*
An attribute name with 1 character is really too short.
33) **Homonymous heterogeneous attributes** *attribute style warning*
Better avoid using the same attribute name with different types on different tables in the same application, as it may confuse the developer.
34) **Mixed table name styles** *schema style notice*
Better use homogeneous table names.
35) **Schema name length quite short** *schema style notice*
A schema name with 4 characters is quite short.
36) **Mixed attribute name styles** *table style notice*
Better use homogeneous attribute names.
37) **Table name length quite short** *table style notice*
A table name with 3 characters is quite short.
38) **Attribute name length quite short** *attribute style notice*
An attribute name of 2 characters is quite short (but "id" and "pk").
39) **Attribute named after its table** *attribute style notice*
An attribute contains the name of its table, which is redundant.
40) **Schema name length short** *schema style information*
A schema name with 5 characters is short.
41) **Table name length short** *table style information*
A table name with 4 characters is short.
42) **Attribute name length short** *attribute style information*
An attribute name with 3 characters is short.
43) **SuperUser with weak password** *user system error*
SuperUser with empty or username password.
44) **Redundant FK** *table system warning*
Redundant Foreign Keys are costly to maintain.
45) **Redundant indexes** *table system warning*
Redundant indexes are costly to maintain.
46) **User with weak password** *user system warning*
User with empty or username password.
47) **Table without index** *table system notice*
Not a single index on a table.